ArgoUML Tutorial, Part 2 of 2

A. O'Riordan, 2009

Contains text from User Manual and other cited sources

The Model

To enable modelling, CASE tools store models. A UML model describes a system using both syntactic element definitions and diagram rendering. The model is the top level model element within ArgoUML. In many respects in ArgoUML it behaves like a package.

ArgoUML is restricted to one model within the tool.

Explorer/*Navigation Perspective*

ArgoUML, like most CASE tools, provides a navigation tree for the designer to access the various parts of the design. ArgoUML provides a rich set of tree-structured perspectives. Within the hierarchical display, elements which have sub-hierarchies are indicated by ⊞ when the hierarchy is hidden and ⊟ when the hierarchy is open.

The designer can choose a navigational perspective from the menu above the navigation tree. There are pre-defined several navigational perspectives that support various tasks in object-oriented software design. Several views of the elements are available such as Package-centric, Diagram-centric, and Class-centric.

Dropping a model element on a diagram is the equivalent of the "Add to Diagram" function. Hence, if the diagram did not yet show this model element, it is added. One can use this drag and drop feature to, for example, easily create a diagram from imported XMI files because XMI files contain model elements, but not any diagram information.

For many systems, a single class diagram will be too complicated to show every association. It is better to create a number of different class diagrams, each concentrating on one aspect of the system. When a class that has been already defined is needed in a second or subsequent diagram, drag its icon from the navigation pane into the new diagram.

The explorer allows the user to view the structure of a model from a number of predefined perspectives. Note that not all model elements are necessarily shown in all perspectives.

- `Package-centric` (the default). The exploring hierarchy is organized by package hierarchy. The top level shows the model. Under this are all the top level packages in the model and all the model elements that are directly in the namespace of the model.

- `Diagram-centric`. In this view the top level comprises all the diagrams in the model. Beneath each diagram is a flat listing of all the model elements on the diagram.

- `Class-centric`. Shows classes in their package hierarchy as well as datatypes and use case diagram elements.

The explorer is designed to be user configurable, to allow the designer to view in his or her preferred way. The "Configure Perspectives" icon ( ) at the top left of the explorer brings up the explorer perspectives dialog.

Button 2 Click over any selected model element in the main area of the explorer will cause a pop-up menu to appear with options such as Copy Diagram to Clipboard as Image, Add to Diagram, Delete From Model.

Model Property Toolbar

`Go up:` Navigate up through the composition structure of the model. Since the model is the top package nothing can happen, and this button is always disabled.

`New Package:` This creates a new Package within the model (which appears on no diagram), navigating immediately to the properties tab for that package.

`New DataType:` This creates a new DataType within the model (which appears on no diagram), navigating immediately to the properties tab for that DataType.

`Delete:` This tool is always disabled, since it is meaningless to delete the model!

Property Fields For The Model

`Name:` The name of the model. The name of a model, like all packages, is by convention all lower case.

Note: The default name supplied to a new model by ArgoUML, `untitledModel`, is thus erroneous and guarantees that ArgoUML always starts up with at least one problem being reported by the design critics.

`Namespace:` Records the namespace for the model. This is the package hierarchy. However since the model is at the top of the hierarchy in ArgoUML, this box is always empty.

Data Types

Standard data types, classes and packages are loaded as sub-packages of the model. These sub-packages are not initially present in the model but are added to the model when used.

Within UML 1.4, `DataType` is a sub-class of the `Classifier` metaclass. It embraces the predefined primitive types (`byte`, `char`, `double`, `float`, `int`, `long` and `short`), the predefined enumeration, `boolean` and user defined *enumeration types*. Within ArgoUML new datatypes may be created using the `New datatype` button on the property tabs of the model and packages.

ArgoUML allows user defined datatypes to be placed on class diagrams to define their inheritence structure. It is represented on the diagram by a box with two compartments, of which the top one is marked with `«datatype»`, and contains the name. The lower one contains operations.

Datatype Property Toolbar

`Go up:`Navigate up through the package structure.

`New datatype:` This creates a new datatype within the same package as the current datatype.

`New Operation:` This creates a new operation within the datatype, navigating immediately to the properties tab for that operation.

`New Stereotype:` This creates a new Stereotype within the same package as the datatype, navigating immediately to the properties tab for that stereotype.

`Delete:`This deletes the datatype from the model.

Property Fields For Datatype

`Name:` The name of the datatype. The primitive datatypes all have lower case names, but there is no formal convention.
`Modifiers:` Has entries `Abstract`, `Leaf` and `Root`. `Abstract` is used to declare that this datatype cannot be instantiated, but must always be specialized. `Leaf` indicates that this datatype can have no further sub-types, while `Root` indicates it is a top level datatype.
`Visibility: Has` entries `public`, `private`, `protected`, and `package`.
`Generalizations:` Lists any datatype that *generalizes* this datatype.
`Specializations`: Lists any specialized datatype.
`Operations`: Lists all the operations defined on this datatype.

Built-in Data Types

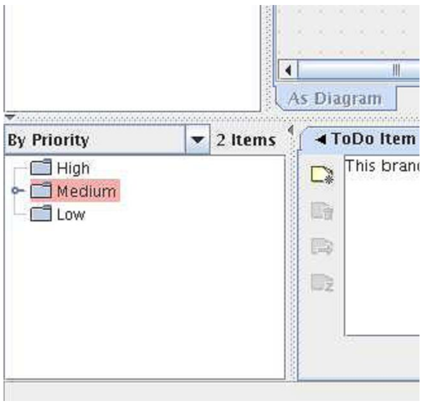
In ArgoUML Java datatypes, classes and interfaces are effectively organized as a hierarchy beneath the overall model. They are grouped in four packages, `lang`, `math`, `net` and `util`, themselves subpackages of `java`, which is a subpackage of the model itself.

You will not find build-in DataTypes, Classes, and Interfaces exposed within the model by default (i.e. they are not present in the explorer). However, once you select one of the built-in DataTypes, Classes, or Interfaces (in the "Type" combo-box on the property sheet of a parameter of an operation of a class), then it becomes visible: you will find that the DataType, Class, or Interface has appeared in the model, and in its correct package stucture for the latter two.

Critics

ArgoUML has processes running in parallel with the design tool, evaluating the current design against models of how “best practice” design might work. These processes are known as *design critics*. ArgoUML is continually monitoring the changes you make to your models and advising you of any problems that they may cause. This criticism of your model is controlled by the Critique pulldown menu. By default the to-do items are organized into three hierarchies by priority: `High`, `Medium` and `Low`.

Recommendations are given in the bottom right pane. This pane provides access to the advice that comes from the Critics processes running within ArgoUML. A selector box at the top allows a choice of how the data is presented (default is priority), a button allows the display of the hierarchy to be changed, and there is an indicator of the number of to-do items identified.



As classes are added more to-do items will appear. The use of *to-do lists* to convey suggestions from the design critics to the user

To-Do items can be added to any component you choose by clicking the To-Do tab in the details pane. A new item is added by clicking on the topmost icon. Items can also be deleted (dismissed), by highlighting the item in the To-Do pane, and then choosing the dustbin icon from the left of the details pane. The relevant “offender” (for example class) will be highlighted in red. They can also be put to sleep (“snooze”).

ArgoUML also provides electronic design checklists as a simple and flexible form of knowledge support. Checklists, like critics, help designers identify design problems early

Menus

Many (but not all) actions that can be carried out on the menu can (and should) be carried out in other ways as well under ArgoUML.

The following are the menus:

- The *File* menu contains operations that affect on the whole project/file.
- The *Edit* menu is generally intended for editing the model or changing the content of a diagram. This menu is not intended for diagram layout functions.
- The *View* menu is for functions that never alter the model, nor the diagram layout, only the way you view the diagram. An example is "zoom". Also navigational functions are here, e.g. "Find" and "Goto Diagram...". All changes of settings in this menu apply to all diagrams (e.g. zoom).
- The *Create* menu contains all possible diagrams that can be created. These functions are context dependent, since they work on the selected model element.
- The *Arrange* menu allows layout changes in the current diagram, which is not the same as the items in the View menu. Functions here can not alter the UML model.
- The *Generation* menu is for Code Generation. The functions here work either on the selected model elements, or on the whole project.
- The *Critique* menu is specific for settings related to critics, which apply for all projects.
- The *Tools* menu is currently empty. If plugins are installed, then their functions appear here.
- The *Help* menu contains the usual "information" and "about".

Settings

This menu item (Edit->Settings) brings up a dialog box, which allows the user to set various options that control the behavior of ArgoUML. These settings are saved persistently for use by subsequent ArgoUML sessions. The information to be saved in put the file `argo.user.properties`. The location of this file is in the `.argouml` directory under the "user’s home directory", which is defined as `${user.home}`

The Profiles tab allows the user to configure the ArgoUML application settings related to the profile. Available profiles are UML 1.4 (default), C++, Java and MetaProfile. Notation tab allows the user to specify certain notation settings, i.e. how things are shown on diagrams. This includes what to show by default, names, visibility, initial values, etc.

Export/Import

ArgoUML saves the diagram information in a PGML file (with extension `.pgml`, the model information in an XMI file (with extension `.xmi` and information about the project in a file with extension `.argo`. All of these are then zipped to a file with extension `.zargo`

Select a diagram, and then navigate to `File=>Export Diagrams`. You can generate GIF, PostScript, Encapsulated PostScript or SVG format.

Beginning with ArgoUML 0.20, XMI 1.2 files are exported conforming to the UML 1.4 metamodel. Select the command `File=>Export as XMI` and choose a filename.

Note about proprietary vendor format - Some tools write non-standard XMI files either by default or always. Here are some tips for getting standard files from a few known tools: Poseidon - Turn **off** the option *Save with diagram data* in the Export Project to XMI dialog.

The Import XMI File option on the Tools or File menu (depending on your version) can be used to import an XMI file from another tool.

Note that ArgoUML can not read an MDL file generated by Rational Rose. MDL is a Rational Rose-specific format. You are much better off getting Rose to export an XMI file.

For now ArgoUML saves diagrams using an earlier proposed standard, *Precision Graphics Markup Language (PGML)*. However it has the option to export graphical data as SVG for those who can make use of it

Tasks

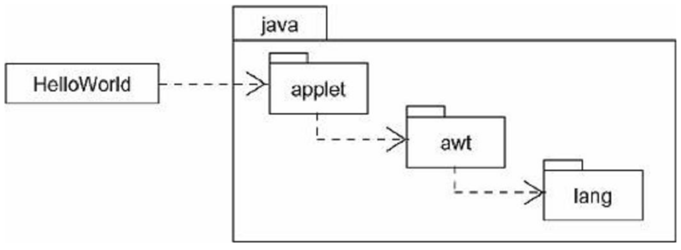
(i) Export the diagrams from last week.

(ii) Export the models as XMI files. Download and run the free community edition of the Poseidon for UML ™ tool (http://www.gentleware.com/) and import the XMI models. Poseidon for UML is a commercial UML modelling tool derived from ArgoUML.

Note: Poseidon for UML (PfU) is delivered in different editions. The Community Edition is the free base version. "It makes learning and using UML a snap and enables the cost-effective exchange of models." [Gentleware Website]. It fully supports UML 2.0. XMI 1.2 is supported as a standard saving format. XMI 1.0, 1.1 and 1.2 can be loaded. Unlike ArgoUML, PfU supports full Undo and Redo.

The Poseidon for UML work area is very similar to ArgoUML including a Navigation Pane, a Drawing Pane, a Details Pane, and an Overview Pane (where the To-Do items were in ArgoUML).

(iii) Draw a system architecture using package notation (Use Cass Diagram in ArgoUML)

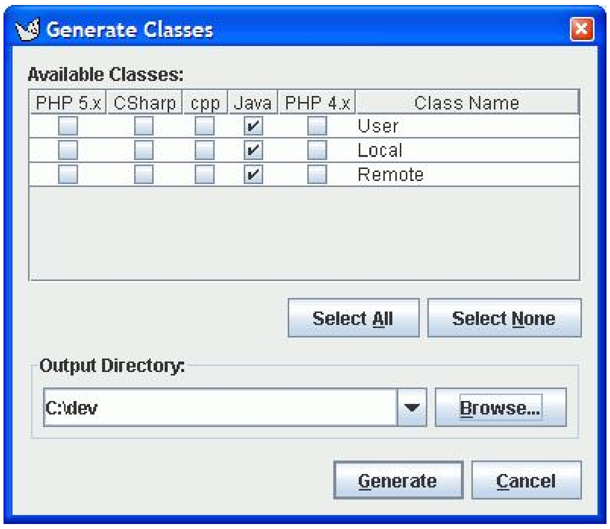


Forward and reverse engineering

Without any plugin modules installed, ArgoUML supports only code generation of Java. ArgoUML V0.20 and later supports the following languages by plugin: C#, C++, php4, and php5.

The current version of ArgoUML will generate a structural template for your code, but is not able to handle behavioral specifications to generate code for the dynamic behavior of the model. Files are generated in a directory hierarchy that need to be filled in by the method implementation code.

ArgoUML's Code Generation Window is displayed when you give a command to generate code files. The table in the upper part of the window lists the classes that will be generated. You can check or uncheck each class to refine the set of classes that will be generated. The text field allows you to specify the output directory where the new class files will be stored.



Generate Classes

Available Classes:

PHP 5.x	CSharp	cpp	Java	PHP 4.x	Class Name
☐	☐	☐	☑	☐	User
☐	☐	☐	☑	☐	Local
☐	☐	☐	☑	☐	Remote

Select All | Select None

Output Directory:

C:\dev

Browse...

Generate | Cancel

The reverse engineering feature is in the menu: "File -> Import Sources...". Reverse engineering in ArgoUML 0.26 can produce Class diagrams and Sequence diagrams (not fully functional).